Projeto 1 - STI

Giovanni Maffeo de Medeiros: 2024232210
Diogo Pinto: 2024156583

Source code: GitHub Repository

Contents

1 Introduction

This project aims to simulate a realistic network scenario to apply our knowledge of information security. The scenario includes a DMZ network, a demilitarized zone designed to expose services to the external network, where are deployed public services such as DNS, WWW and SMTP. Additionally, it features a LAN network, a local area network intended for internal communication and access to private resources, where are hosted internal services like FTP and client machines.

Both networks are connected through a Linux router acting as a border gateway. How we learn, this router has two important roles in this scenario. The first is to isolate ensures isolation between networks (DMZ, LAN and Internet). It is important because this networks have different levels of public access, ike LAN is just for private access, while DMZ has some services openned to Internet. Therefore, all the packages should pass throw the router and he will be responsable to redirect to the certain network performing packet forwarding.

As we learned, alem of routing packages, another essencial role for this type of router is garantee security, so this it is in this router that we configure firewall rules to control the level of access to the services, which the default policy is to negate every access, and Suricata to protect our services from external threats.

This project aims to simulate a realistic network scenario in order to apply our knowledge of information security. The scenario includes a DMZ (Demilitarized Zone), a network segment designed to expose services to external access, where public services such as DNS, WWW and SMTP are deployed. In addition, it features a LAN (Local Area Network), intended for internal communication and access to private resources, where services like FTP and client machines are hosted.

Both networks are connected through a Linux router acting as a border gateway. As we have learned, this type of router plays two key roles. The first is to ensure isolation between networks (DMZ, LAN and Internet), which is essential since each network has a different level of exposure, for example, the LAN is restricted to private access, while the DMZ includes services accessible from the Internet. Therefore, all traffic must go through the router, which is responsible for forwarding packets to the appropriate destination.

Besides routing, the second essential role of this type of router is to guarantee network security. It is on this gateway that we configure firewall rules to control access to services (with the default policy being to deny all access) and deploy Suricata to detect and prevent external threats.

2 Scenario description

To simulate the proposed scenario, we created three Kali virtual machines to host the LAN network and its services, another virtual machine to host the DMZ and its respective services and a separate VM to act as the Linux router. To simulate the Internet and the external services, we decided to use the host operating system , where the virtual machines are running, in this case, Windows. Then, to interconnect the environments, we created three VirtualBox Host-Only Ethernet Adapter networks using Oracle VirtualBox, as described below.

1. The first network connects the DMZ to the Linux router and uses the address range `192.168.56.0/24`.
2. The second network connects the LAN to the Linux router and uses the range `192.168.205.0/24`.
3. The third network simulates the Internet and its public services. Since we needed to host simulated external services, we created this third network with the range `192.168.52.0/24`, which connects the host operating system (in our case, Windows, acting as the Internet) to the Linux router.

Following the description above, we present the figure from the project assignment provided by the professors, adapted to our scenario, that is, including the IP addresses of the services and the network ranges we chose.

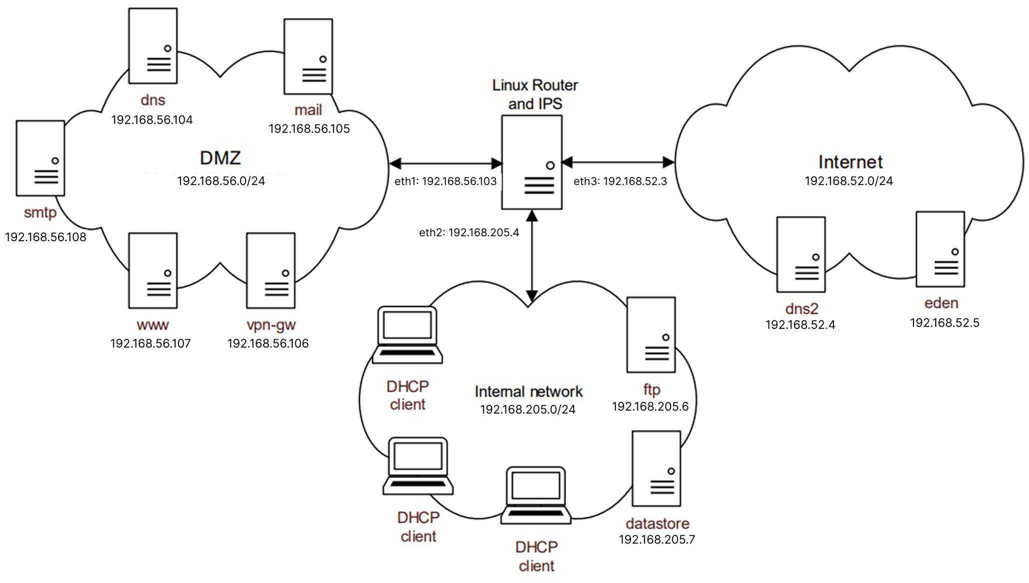


Figure 1: Scenario description

In addition to the figure, to support the understanding of the scenario, we also present the tables below, which describe the IP addresses assigned to each machine and network range.

Interface	**Connected Network**	**IP Address**
eth1	DMZ	192.168.56.103
eth2	LAN	192.168.205.4
eth3	Internet / Host (Windows)	192.168.52.3

Table 1: Linux router interfaces

DMZ network (192.168.56.0/24) eth0: 192.168.56.102	
Service	**IP Address**
dns	192.168.56.104
mail	192.168.56.105
vpn-gw	192.168.56.106
www	192.168.56.107
smtp	192.168.56.108

Table 2: DMZ network and services

LAN network (192.168.205.0/24) eth0: 192.168.205.5	
Service	**IP Address**
ftp	192.168.205.6
datastore	192.168.205.7

Table 3: LAN network and services

Internet network (192.168.52.0/24) Ethernet 4: 192.168.52.1	
Service	**IP Address**
dns2	192.168.52.4
eden	192.168.52.5
external_ftp	192.168.52.6

Table 4: Internet network and simulated external services

2.1 Hostname and IP mapping

To simplify the setup and the configuration of the rules , since we would frequently need to use the IP addresses, we edited the name resolution mapping files to associate IPs with service names on each virtual machine and on the host operating system. On the virtual machines, we modified the `/etc/hosts` file and, on Windows, we updated the `System32\drivers\etc\hosts` file. An example of the configuration used in the DMZ machine is shown below, following the IP addresses defined in Section 2.

```
192.168.56.103  linux_router
192.168.205.5   lan
192.168.56.104  dns
192.168.56.105  mail
192.168.56.106  vpn-gw
192.168.56.107  www
192.168.56.108  smtp
192.168.205.6   ftp
192.168.52.4    dns2
192.168.52.5    eden
192.168.205.7   datastore
192.168.52.6    external_ftp
```

It is important to note that the IP address of the Linux router varies for each machine, as it must correspond to the network to which it is connected. For example, the DMZ VM uses the router's IP address `192.168.56.103`, while the LAN VM uses `192.168.205.4`. All other services and virtual machines communicate using their own IP addresses, since the Linux router is responsible for forwarding packets between networks.

2.2 Routing setup for simulated services

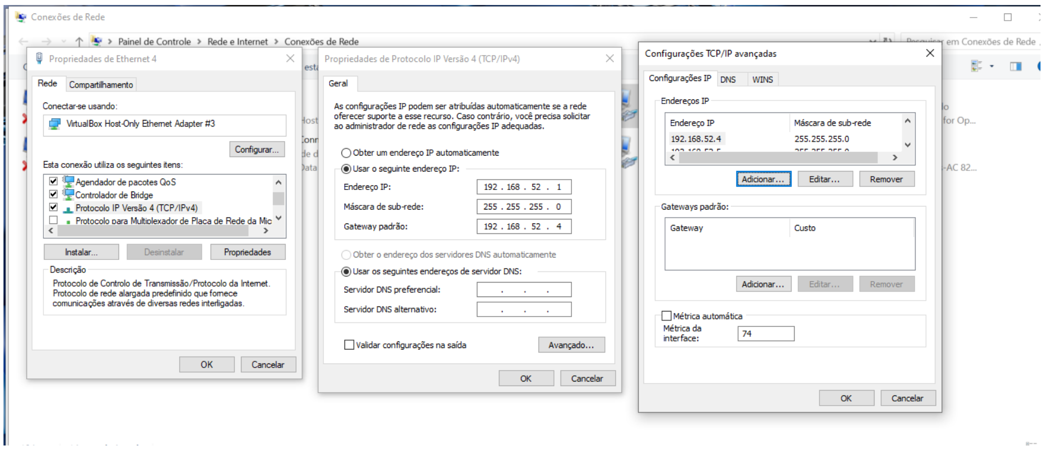
In addition to name resolution mapping, we needed to simulate that each VM is hosting the services described in Section 2. To achieve this, we added secondary IP addresses to the same interface that connects to the Linux router, where each of these secondary IPs simulates a different service hosted on the VM.

For the Kali VMs, we edited the `/etc/network/interfaces` file to assign the additional IP addresses to the `eth0` interface. After that, we applied the configuration using `sudo service networking restart`. To view all IP addresses currently assigned to `eth0`, we can use the command `ip addr show eth0`. An example of the configuration used in the DMZ machine is shown below.

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
```

```
iface eth0 inet static
  address 192.168.56.102
  netmask 255.255.255.0
  post-up ip addr add 192.168.56.104/24 dev eth0
  post-up ip addr add 192.168.56.105/24 dev eth0
  post-up ip addr add 192.168.56.106/24 dev eth0
  post-up ip addr add 192.168.56.107/24 dev eth0
  post-up ip addr add 192.168.56.108/24 dev eth0
```

To add secondary IP addresses on Windows permanently, we need to navigate through: *Control Panel > Network and Sharing Center > Change adapter settings > Properties > Internet Protocol Version 4 (TCP/IPv4) > Advanced*, as shown in the image below.



Conectar-se usando: VirtualBox Host-Only Ethernet Adapter #3

Esta conexão utiliza os seguintes itens:
- [✓] Agendador de pacotes QoS
- [✓] Controlador de Bridge
- [✓] Protocolo IP Versão 4 (TCP/IPv4)
- [ ] Protocolo para Multiplexador de Placa de Rede da Mic

Descrição: Protocolo de Controlo de Transmissão/Protocolo da Internet. Protocolo de rede alargada predefinido que fornece comunicações através de diversas redes interligadas.

As configurações IP podem ser atribuídas automaticamente se a rede oferecer suporte a esse recurso. Caso contrário, você precisa solicitar ao administrador de rede as configurações IP adequadas.

Obter um endereço IP automaticamente [✓ Usar o seguinte endereço IP:]

Endereço IP:	192 . 168 . 52 . 1
Máscara de sub-rede:	255 . 255 . 255 . 0
Gateway padrão:	192 . 168 . 52 . 4

Obter o endereço dos servidores DNS automaticamente [✓ Usar os seguintes endereços de servidor DNS:]

Servidor DNS preferencial:	. . .
Servidor DNS alternativo:	. . .

[ ] Validar configurações na saída

Endereços IP

Endereço IP	Máscara de sub-rede
192.168.52.4	255.255.255.0

Gateways padrão:

Gateway	Custo

[ ] Métrica automática

Métrica da interface: 74

Figure 2: Secondary IPs Windows

To view the IP address mappings assigned to the network interface, we can use the following command: *ipconfig* and look for Ethernet 4.

3 Routing and connectivity configuration

In this section, we explain how to configure connectivity between the networks through the Linux router.

3.1 Routing by linux router

To enable routing between networks, the Linux router must be configured to forward IP packets. This is done by enabling the `ip_forward` setting in the system configuration.

```
# Edit the system configuration file
sudo nano /etc/sysctl.conf
```

```
# Add the following line to enable IP forwarding
net.ipv4.ip_forward = 1

# Apply the changes immediately
sudo sysctl -p

# Confirm that IP forwarding is active
sudo cat /proc/sys/net/ipv4/ip_forward
```

It is important to note that the command `sudo sysctl -p` must be executed again after each reboot in order to apply the routing configuration.

3.2 Default gateway

Although IP routing is enabled on the Linux router, routing between machines will not yet work at this stage. This is because, while the router is configured to forward packets, the other machines are not yet sending their outgoing traffic through it. In order to make the Linux router effectively forward packets, it must be defined as the default gateway on each connected machine.

For this purpose, on the Kali Linux virtual machines, we edited the `/etc/network/interfaces` file to define the Linux router as the gateway for the interface connected to it (`eth0`). These configuration lines indicate that all outgoing traffic should be redirected through the Linux router. It is important to note that the IP address of the Linux router varies for each machine, as it must correspond to the network to which it is connected as described on Subsection 2.1, as shown below:

```
# DMZ machine:
sudo nano /etc/network/interfaces
auto eth0
iface eth0 inet static
  address 192.168.56.102
  netmask 255.255.255.0
  gateway 192.168.56.103
  ...

# LAN machine:
sudo nano /etc/network/interfaces
auto eth0
iface eth0 inet static
  address 192.168.205.5
  netmask 255.255.255.0
  gateway 192.168.205.4
  ...
```

For the Windows host, which simulates the Internet and external services, the configuration is slightly different and to make these routes persistent across reboots, we use the `-p` option. The configuration is shown below:

```
# Windows / Internet machine:
route -p add 192.168.56.0 mask 255.255.255.0 192.168.52.3
route -p add 192.168.205.0 mask 255.255.255.0 192.168.52.3

# To verify the routes:
route print
```

After these configurations, whenever a machine tries to access an IP address that is not part of its own subnet, the packet will be routed to the default gateway (the Linux router). If the router has a route to reach the destination IP, it will forward the packet accordingly.

3.3 Tests

Before setting the default gateway on the machines, any attempt to reach IP addresses outside the local subnet fails, as there is no route for forwarding packets through the Linux router. The examples below illustrate this behavior:

```
# From the DMZ machine to the LAN:
kali@dmz:~$ ping lan

ping: connect: Network is unreachable

# From the LAN machine to the DMZ:
kali@lan:~$ ping dmz

ping: connect: Network is unreachable
```

After defining the default gateway on each machine, we tested the connectivity between the networks. The following outputs show successful ICMP responses (ping) sent to the Linux router interface, confirming that routing is working as expected.

```
# From DMZ to LAN
( kalikali)-[~]
$ ping lan

PING lan (192.168.205.4) 56(84) bytes of data.
64 bytes from 192.168.205.4: icmp_seq=1 ttl=64 time=2.85 ms
^C
--- lan ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2.852/2.852/2.852/0.000 ms

# From LAN to DMZ
( kalikali)-[~]
$ ping dmz
PING dmz (192.168.56.102) 56(84) bytes of data.
64 bytes from 192.168.56.102: icmp_seq=1 ttl=63 time=2.78 ms
^C
--- dmz ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2.779/2.779/2.779/0.000 ms
```

4 Firewall configuration to protect the router

In this section, we configure firewall rules on the Linux router to protect it from unauthorized access. Since the router is directly connected to all networks, the rules defined here control incoming traffic to the router itself, that is the `INPUT` chain.

4.1 Default input policy

The default policy for the `INPUT` chain defines what happens to incoming packets that do not match any specific rule. By setting this policy to `DROP`, we deny all unsolicited traffic by default.

The following command sets this policy:

```
sudo iptables -P INPUT DROP
```

After applying this rule, any incoming packet that is not explicitly accepted will be dropped. The test below demonstrates that:

```
# VM LAN: trying to ping the router (192.168.205.4)
( kalikali)-[~]
$ ping linux_router
PING 192.168.205.4 (192.168.205.4) 56(84) bytes of data.
^C
--- 192.168.205.4 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1010ms

# Linux router: checking the iptables configuration
( kalikali)-[~]
$ sudo iptables -L -v -n

Chain INPUT (policy DROP 11 packets, 1776 bytes)
 pkts bytes target     prot opt in     out     source               destination
...
```

4.2 DNS name resolution requests sent to outside servers

To allow the Linux router to resolve domain names using external DNS servers, we must permit outgoing DNS requests and their responses. We did not apply policies for `OUTPUT` chain, so we need to allow responses of any source. Since may use either UDP or TCP, both protocols can be allowed in the `INPUT` chain:

```
sudo iptables -A INPUT -p udp --sport 53 -j ACCEPT
sudo iptables -A INPUT -p tcp --sport 53 -j ACCEPT
```

To validate this configuration, we tested DNS queries directly from the Linux router to a global public server. Since the Linux router is also connected to the real Internet and not only to the Windows network created to simulate the Internet. However, the behavior would be the same for `dns2` or any other server, as the rule does not restrict the source.

```
# From Linux router with udp:
( kalikali)-[~]
$ dig @8.8.8.8 google.com

; <<>> DiG 9.20.2-1-Debian <<>> @8.8.8.8 google.com
...
;; SERVER: 8.8.8.8#53(8.8.8.8) (UDP)
...

# From Linux router with tcp:
( kalikali)-[~]
$ dig +tcp @8.8.8.8 google.com
; <<>> DiG 9.20.2-1-Debian <<>> +tcp @8.8.8.8 google.com
...
;; SERVER: 8.8.8.8#53(8.8.8.8) (TCP)
...

# Linux router rules:
( kalikali)-[~]
$ sudo iptables -L -v -n

Chain INPUT (policy DROP 132 packets, 26164 bytes)
 pkts bytes target     prot opt in     out     source               destination
...
    1    83 ACCEPT     udp  --  *      *       0.0.0.0/0            0.0.0.0/0
                 udp spt:53
    5   325 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0
                 tcp spt:53
```

4.3 SSH connections to the router system

First, we started and enabled the SSH service on the router:

```
# Enable and start the SSH service on the Linux router
sudo systemctl enable ssh
sudo systemctl start sshd.service
```

Then, we created a firewall rule to allow incoming TCP connections on port 22, which corresponds to a SSH connection, from the LAN subnet and from vpn-gw.

```
sudo iptables -A INPUT -p tcp --dport 22 -s 192.168.205.0/24 -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 22 -s vpn-gw -j ACCEPT
```

```
# From a LAN:
( kalikali)-[~]
$ telnet 192.168.205.4 22
Trying 192.168.205.4...
Connected to 192.168.205.4.
Escape character is '^]'.
SSH-2.0-OpenSSH_9.9p1 Debian-3

# From vpn-gw:
( kalikali)-[~]
```

```
$ nc -s vpn-gw linux_router 22
SSH-2.0-OpenSSH_9.9p1 Debian-3

# Linux router:
Chain INPUT (policy DROP 12 packets, 2304 bytes)
 pkts bytes target     prot opt in     out     source               destination
    4   221 ACCEPT     tcp  --  *      *       192.168.205.0/24     0.0.0.0/0
                 tcp dpt:22
    3   164 ACCEPT     tcp  --  *      *       192.168.56.106       0.0.0.0/0
                 tcp dpt:22
...
```

5 Firewall configuration to authorize direct communications (without NAT)

In this section, we configure firewall rules on the Linux router to allow direct communication between machines in different networks without using NAT. Since we need to control access between the networks and, as previously explained, any packet exchanged between them is forwarded through the Linux router, these rules must be configured in the `FORWARD` chain.

5.1 Default forward policy

The default policy for the `FORWARD` chain defines how packets that need to be routed between networks are handled. By setting this policy to `DROP`, we deny all traffic by default.

```
sudo iptables -P FORWARD DROP
```

5.2 Domain name resolutions using the dns server

As the DNS server must be responsible for name resolution regardless of any source network, we used a generic rule without restricting the source IP:

```
sudo iptables -A FORWARD -p udp -d dns --dport 53 -j ACCEPT
```

To verify the rule, we executed the test below:

```
# From VM LAN:
( kalikali)-[~]
$ echo "hello world" | nc -u dns 53

# VM DMZ:
( kalikali)-[~]
$ nc -ulp 53

hello world

# Linux router rules:
...
```

```
Chain FORWARD (policy DROP 1 packets, 40 bytes)
 pkts bytes target     prot opt in     out     source               destination
    1    40 ACCEPT     udp  --  *      *       0.0.0.0/0
        192.168.56.104       udp dpt:53
...
```

5.3 External DNS resolution via the dns server

To allow the DNS server in the DMZ to resolve names using external DNS servers, such as `dns2` or Internet-based servers, we created a rule that permits outgoing UDP DNS queries from the `dns` server without restricting the destination.

```
sudo iptables -A FORWARD -s dns -p udp --dport 53 -j ACCEPT
```

For the tests, we also needed to download Nmap on Windows:

```
https://nmap.org/download.html
```

To verify the rule, we executed the test below:

```
# From VM DMZ:
( kalikali)-[~]
$ echo "pkt" | nc -u -s dns internet 53

# Windows (Internet simulation):
C:\Windows\system32>ncat.exe -ul 53
pkt

# Linux router rules:
...
Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
 ...
    1    40 ACCEPT     udp  --  *      *       192.168.56.104       0.0.0.0/0
                   udp dpt:53
...
```

5.4 DNS zone synchronization between dns and dns2

To allow DNS zone synchronization between the `dns` and `dns2` servers, we needed to configure specific rules that permit TCP communication on port 53 in both directions. Additionally, we included a rule to allow the forwarding of responses and related packets, which is essential for the proper completion of TCP-based exchanges between the two servers.

```
sudo iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
sudo iptables -A FORWARD -s dns2 -d dns -p tcp --dport 53 -j ACCEPT
sudo iptables -A FORWARD -s dns -d dns2 -p tcp --dport 53 -j ACCEPT
```

To verify the rule, we executed the test below, which show dns to dns2:

```
# From VM DMZ (dns):
( kalikali)-[~]
$ echo "pkt" | nc -s dns dns2 53

# Windows (dns2):
C:\Windows\system32>ncat.exe -l 53
pkt

# Linux router rules:
Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
...
    4   176 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0
                   state RELATED,ESTABLISHED
...
    1    60 ACCEPT     tcp  --  *      *       192.168.56.104       192.168.52.4
                tcp dpt:53
...
```

Another test we executed was from dns2 to dns:

```
# From VM Windows (dns2):
C:\Windows\system32>echo pkt | ncat.exe --source dns2 dns 53

# VM DMZ (dns):
( kalikali)-[~]
$ nc -lp 53
pkt

# Linux router rules:
Chain FORWARD (policy DROP 4 packets, 208 bytes)
 pkts bytes target     prot opt in     out     source               destination
...
    2   104 ACCEPT     tcp  --  *      *       192.168.52.4
        192.168.56.104       tcp dpt:53
...
```

5.5 SMTP connections to the smtp server

To allow machines any source to send emails through the `smtp` server, we added a rule permitting TCP connections to port 25 on destined to that server.

```
sudo iptables -A FORWARD -d smtp -p tcp --dport 25 -j ACCEPT
```

To verify the rule, we executed the test below:

```
# From VM LAN:
( kalikali)-[~]
$ echo "pkt" | nc smtp 25

# VM DMZ (smtp):
( kalikali)-[~]
$ nc -lp 25
pkt

# Linux router rules:
```

```
Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
...
    1    60 ACCEPT     tcp  --  *      *       0.0.0.0/0
        192.168.56.108       tcp dpt:25
```

5.6 POP and IMAP connections to the mail server

To allow clients from the internal network to retrieve emails from the `mail` server using the POP protocol, we added a rule permitting TCP traffic to port 110.

```
sudo iptables -A FORWARD -d mail -p tcp --dport 110 -j ACCEPT
```

To verify the rule, we executed the test below:

```
# From VM LAN:
( kalikali)-[~]
$ echo "pkt" | nc mail 110

# VM DMZ (mail):
( kalikali)-[~]
$ nc -lp 110
pkt

# Linux router rules:
Chain FORWARD (policy DROP 9 packets, 540 bytes)
 pkts bytes target     prot opt in     out     source               destination
...
    1    60 ACCEPT     tcp  --  *      *       0.0.0.0/0
        192.168.56.105       tcp dpt:110
```

We also added a rule for TCP traffic to port 143 to IMAP connections.

```
sudo iptables -A FORWARD -d mail -p tcp --dport 143 -j ACCEPT
```

To verify the rule, we executed the test below:

```
# From VM LAN:
( kalikali)-[~]
$ echo "pkt" | nc mail 143

# VM DMZ (mail):
( kalikali)-[~]
$ nc -lp 143
pkt

# Linux router rules:
Chain FORWARD (policy DROP 9 packets, 540 bytes)
 pkts bytes target     prot opt in     out     source               destination
...
    1    60 ACCEPT     tcp  --  *      *       0.0.0.0/0
        192.168.56.105       tcp dpt:143
```

5.7 HTTP and HTTPS connections to the www server

To allow HTTP traffic to the `www` server located in the DMZ, we created a rule allowing TCP packets on port 80 for HTTP connections and on port 443 for HTTPS connections.

```
sudo iptables -A FORWARD -d www -p tcp --dport 80 -j ACCEPT
```

To verify the rule, we executed the test below:

```
# From VM LAN:
( kalikali)-[~]
$ echo "pkt" | nc www 80

# VM DMZ (www):
( kalikali)-[~]
$ nc -lp 80
pkt

# From VM LAN:
( kalikali)-[~]
$ echo "pkt" | nc www 443

# VM DMZ (www):
( kalikali)-[~]
$ nc -lp 443
pkt

# Linux router rules:
Chain FORWARD (policy DROP 9 packets, 540 bytes)
 pkts bytes target     prot opt in     out     source               destination
...
    1    60 ACCEPT     tcp  --  *      *       0.0.0.0/0
        192.168.56.107       tcp dpt:80
    1    60 ACCEPT     tcp  --  *      *       0.0.0.0/0
        192.168.56.107       tcp dpt:443
```

5.8 OpenVPN connections to the vpn-gw server

To enable VPN connectivity through the `vpn-gw` server in the DMZ, we added two rules allowing TCP and UDP traffic to port 1194, which is the default port used by OpenVPN.

```
sudo iptables -A FORWARD -d vpn-gw -p udp --dport 1194 -j ACCEPT
sudo iptables -A FORWARD -d vpn-gw -p tcp --dport 1194 -j ACCEPT
```

To verify the rule, we executed the test below:

```
# From VM LAN:
( kalikali)-[~]
$ echo "pkt" | nc -u vpn-gw 1194

# VM DMZ (vpn-gw):
( kalikali)-[~]
$ nc -ulp 1194
pkt
```

```
# Linux router rules:
Chain FORWARD (policy DROP 9 packets, 540 bytes)
 pkts bytes target     prot opt in     out     source               destination
...
    1    32 ACCEPT     udp  --  *      *       0.0.0.0/0
        192.168.56.106       udp dpt:1194
    0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0
        192.168.56.106       tcp dpt:1194
```

5.9 Internal network access from VPN clients via vpn-gw

When a remote VPN client connects to the `vpn-gw` server and tries to access resources in the internal network (LAN), the `vpn-gw` performs Source NAT (SNAT), replacing the original source IP address with its own. This causes the devices on the LAN to see the connection as if it originated directly from `vpn-gw` and send the response directly to it.

To allow such communication, we configured the following rule on the Linux router, permitting any type of packet and protocol from `vpn-gw` to the LAN:

```
sudo iptables -A FORWARD -s vpn-gw -d 192.168.205.0/24 -j ACCEPT
```

To verify the rule, we executed the test below:

```
# From VM DMZ using any IP:
( kalikali)-[~]
$ ping lan
PING lan (192.168.205.5) 56(84) bytes of data.
^C
--- lan ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3048ms

# From VM DMZ specifying source IP of vpn-gw:
( kalikali)-[~]
$ ping -I 192.168.56.106 lan
PING lan (192.168.205.5) from 192.168.56.106 : 56(84) bytes of data.
64 bytes from lan (192.168.205.5): icmp_seq=1 ttl=63 time=2.96 ms
64 bytes from lan (192.168.205.5): icmp_seq=2 ttl=63 time=2.68 ms
64 bytes from lan (192.168.205.5): icmp_seq=3 ttl=63 time=2.14 ms
64 bytes from lan (192.168.205.5): icmp_seq=4 ttl=63 time=2.63 ms
64 bytes from lan (192.168.205.5): icmp_seq=5 ttl=63 time=3.04 ms
64 bytes from lan (192.168.205.5): icmp_seq=6 ttl=63 time=2.41 ms
^C
--- lan ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5009ms
rtt min/avg/max/mdev = 2.142/2.643/3.041/0.306 ms

# Linux router rules:
Chain FORWARD (policy DROP 13 packets, 876 bytes)
...
    1    84 ACCEPT     all  --  *      *       192.168.56.106
        192.168.205.0/24
...
```

6 Firewall configuration for connections to the external IP address of the firewall (using NAT)

In this section, we explore the connections initiated from external hosts towards the ”public” interface of the Linux router, that is, the one connected to the Windows network. These connections target internal services located in the DMZ or LAN networks.

To achieve this, Destination NAT (DNAT) is used in the `PREROUTING` chain. This mechanism allows the firewall to intercept incoming packets and modify their destination IP address before routing decisions are made, redirecting them to the appropriate internal machine hosting the intended service. Additionally, the firewall must be configured to allow these forwarded packets, as long as corresponding to authorized connections.

6.1 FTP connections (passive and active modes) to the ftp server

This section is divided in two parts: FTP server installation / configuration and FTP firewall rules using NAT.

6.1.1 FTP server configuration

To begin, the FTP server must be installed on the LAN virtual machine. Since the system initially has no access to the Internet, we temporarily connect the VM using a `Bridge Adapter` to allow external access for package installation.

Then, on the LAN VM, we proceed with the installation and setup of the `vsftpd` server as follows:

1. Comment out the default gateway in the `/etc/network/interfaces` file:

```
auto eth0
iface eth0 inet static
  address 192.168.205.5
  netmask 255.255.255.0
  # gateway 192.168.205.4
```

2. Restart the networking service:

```
sudo systemctl restart networking
```

3. Update package lists and install `vsftpd`:

```
sudo apt update
sudo apt install vsftpd
```

4. Start the FTP server:

```
sudo systemctl start vsftpd
```

5. Finally, restore the original network configuration by uncommenting the default gateway and restarting the network service again.

6. Configure the FTP server to listen on the IP address assigned to the FTP service according to Subsection 2.1. Edit the configuration file:

```
sudo nano /etc/vsftpd.conf

listen=YES
listen_ipv6=NO
listen_address=192.168.205.6
```

7. Limit the range of ports used in passive mode, since the default behavior is to select random ports. We need to define a fixed range to properly configure the firewall forwarding rules. Edit the same file again:

```
sudo nano /etc/vsftpd.conf

pasv_enable=YES
pasv_min_port=30000
pasv_max_port=30100
```

8. Then restart the FTP service:

```
sudo systemctl restart vsftpd
```

6.1.2 *FTP server firewall rules*

To allow access to the FTP server located in the internal LAN network from the external network, we configured both DNAT and FORWARD rules in the Linux router.

We redirected incoming TCP connections from the external interface (`eth3` connected to Windows network) to the internal FTP server's IP. The rules below handle both active mode (port 21) and passive mode (ports `30000:30100`):

```
# DNAT rules (PREROUTING)
sudo iptables -t nat -A PREROUTING -i eth3 -p tcp --dport 21 -j DNAT --to-
    destination 192.168.205.6
sudo iptables -t nat -A PREROUTING -i eth3 -p tcp --dport 30000:30100 -j DNAT --
    to-destination 192.168.205.6

# FORWARD rules
sudo iptables -A FORWARD -p tcp -d ftp --dport 21 -j ACCEPT
sudo iptables -A FORWARD -p tcp -d ftp --dport 30000:30100 -j ACCEPT
```

To verify the rules, we executed the test below from a Windows host using `ncftp`, which supports passive FTP connections:

```
# On Windows (external):
C:\Windows\system32> ncftp -u kali linux_router

NcFTP 3.2.6 (Nov 15, 2016) by Mike Gleason (http://www.NcFTP.com/contact/).
Resolving linux_router...
Connecting to 192.168.52.3...

(vsFTPd 3.0.5)
Logging in...
Password requested by 192.168.52.3 for user "kali".
Password: ****

Login successful.
Logged in to linux_router.

ncftp /home/kali > ls
Desktop/     Documents/   Downloads/   Music/       Pictures/    Public/
    Templates/   Videos/
ncftp /home/kali > bye
```

After testing, we inspected the iptables rules and confirmed the PREROUTING (DNAT) and FORWARD chains registered the connections as expected:

```
# Linux router:
( kalikali)-[~]
$ sudo iptables -L -v -n
$ sudo iptables -t nat -L -v -n

Chain FORWARD (policy DROP)
 pkts bytes target     prot opt in     out     source     destination
...
    1    52 ACCEPT     tcp  --  *      *       0.0.0.0/0  192.168.205.6  tcp dpt
       :21
    1    52 ACCEPT     tcp  --  *      *       0.0.0.0/0  192.168.205.6  tcp dpts
       :30000:30100
...

Chain PREROUTING (policy ACCEPT)
 pkts bytes target     prot opt in     out     source     destination
    1    52 DNAT       tcp  --  eth3   *       0.0.0.0/0  0.0.0.0/0       tcp dpt
       :21 to:192.168.205.6
    1    52 DNAT       tcp  --  eth3   *       0.0.0.0/0  0.0.0.0/0       tcp
       dpts:30000:30100 to:192.168.205.6
```

To confirm that the passive mode test works, we removed the configuration that fixed the port range on the FTP server and observed that passive mode connections stopped working.

```
C:\Windows\system32>ncftp -u kali linux_router
NcFTP 3.2.6 (Nov 15, 2016) by Mike Gleason (http://www.NcFTP.com/contact/).
Resolving linux_router...
Connecting to 192.168.52.3...

(vsFTPd 3.0.5)
Logging in...
```

```
Password requested by 192.168.52.3 for user "kali".

    Please specify the password.

Password: ****

Login successful.
Logged in to linux_router.

ncftp /home/kali > ls
Data connection to 192.168.205.6:27958 timed out.
```

6.2 SSH connections to the datastore server from eden and dns2 only

First, we should stop the SSH service on the `LAN (datastore)` server to allow testing with `netcat` using port 22. This can be done by executing the following command:

```
sudo systemctl stop ssh
```

The firewall must only allow SSH access to the `datastore` server from the hosts `eden` and `dns2` using `PREROUTING`. Therefore, we added the following rules to the Linux router:

```
sudo iptables -t nat -A PREROUTING -i eth3 -p tcp -s eden --dport 22 -j DNAT --to
    -destination 192.168.205.7:22
sudo iptables -t nat -A PREROUTING -i eth3 -p tcp -s dns2 --dport 22 -j DNAT --to
    -destination 192.168.205.7:22

sudo iptables -A FORWARD -s eden -d datastore -p tcp --dport 22 -j ACCEPT
sudo iptables -A FORWARD -s dns2 -d datastore -p tcp --dport 22 -j ACCEPT
```

To verify the rule, we executed the test below:

```
# From Windows (allowed sources):
C:\Windows\system32>echo pkt | ncat.exe --source dns2 datastore 22
C:\Windows\system32>echo pkt2 | ncat.exe --source eden datastore 22

# From Windows (disallowed source):
C:\Windows\system32>echo pkt | ncat.exe datastore 22
Ncat: TIMEOUT.

# On LAN VM:
( kalikali)-[~]
$ nc -lp 22
pkt
( kalikali)-[~]
$ nc -lp 22
pkt2

# Linux router rules:
( kalikali)-[~]
$ sudo iptables -L -v -n
sudo iptables -t nat -L -n -v

```

```
Chain FORWARD (policy DROP 15 packets, 780 bytes)
 pkts bytes target     prot opt in     out     source               destination
...
    1    52 ACCEPT     tcp  --  *      *       192.168.52.5         192.168.205.7
                tcp dpt:22
    2   104 ACCEPT     tcp  --  *      *       192.168.52.4         192.168.205.7
                tcp dpt:22

Chain PREROUTING (policy ACCEPT 66 packets, 5964 bytes)
 pkts bytes target     prot opt in     out     source               destination
...
    1    52 DNAT       tcp  --  eth3   *       192.168.52.5         0.0.0.0/0
                   tcp dpt:22 to:192.168.205.7:22
    2   104 DNAT       tcp  --  eth3   *       192.168.52.4         0.0.0.0/0
                   tcp dpt:22 to:192.168.205.7:22
```

7 Firewall configuration for communications from the internal network to the outside (using NAT)

In this section, we configure the firewall to allow internal machines to initiate outbound connections to external networks, such as the Internet. These connections must be routed through the Linux router, which will apply Network Address Translation (NAT) using the `MASQUERADE` target. This operation replaces the source IP address of the outgoing packets with the external IP address of the router.

In addition to NAT, forwarding must be explicitly allowed by adding rules to the `FORWARD` chain to authorize traffic originating from the internal network and destined to the Internet.

7.1 DNS resolutions from internal network to the Internet

To allow machines in the LAN network to make DNS requests to external servers (e.g., `dns2`), the router must forward the packets through `eth3` and apply NAT using `MASQUERADE` to change the source IP address to the external IP of the router.

Linux router rule:

```
sudo iptables -A FORWARD -s 192.168.205.0/24 -p udp --dport 53 -o eth3 -j ACCEPT
sudo iptables -t nat -A POSTROUTING -s 192.168.205.0/24 -p udp --dport 53 -o eth3
     -j MASQUERADE
```

To verify the rule, we executed the test below:

```
# From VM LAN:
( kalikali)-[~]
$ echo "pkt" | nc -u dns2 53

# On Windows:
C:\Windows\system32>ncat.exe -ul 53
```

```
pkt

# Linux router rules:
...
Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out    source           destination
...
    1    32 ACCEPT     udp  --  *      eth3   192.168.205.0/24  0.0.0.0/0   udp
        dpt:53

Chain POSTROUTING (policy ACCEPT 37 packets, 11192 bytes)
 pkts bytes target       prot opt in   out    source           destination
 ...
    1    32 MASQUERADE   udp  --  *    eth3   192.168.205.0/24  0.0.0.0/0   udp
        dpt:53
```

7.2 HTTP, HTTPS and SSH connections from internal network

To allow machines in the LAN network to access external services over HTTP (port 80), HTTPS (port 443), and SSH (port 22), we must authorize the forwarding of TCP packets through `eth3` and apply NAT using `MASQUERADE` to modify the source IP.

Linux router rules:

```
sudo iptables -A FORWARD -s 192.168.205.0/24 -p tcp --dport 80 -o eth3 -j ACCEPT
sudo iptables -A FORWARD -s 192.168.205.0/24 -p tcp --dport 443 -o eth3 -j ACCEPT
sudo iptables -A FORWARD -s 192.168.205.0/24 -p tcp --dport 22 -o eth3 -j ACCEPT

sudo iptables -t nat -A POSTROUTING -s 192.168.205.0/24 -p tcp --dport 80 -o eth3
     -j MASQUERADE
sudo iptables -t nat -A POSTROUTING -s 192.168.205.0/24 -p tcp --dport 443 -o
    eth3 -j MASQUERADE
sudo iptables -t nat -A POSTROUTING -s 192.168.205.0/24 -p tcp --dport 22 -o eth3
     -j MASQUERADE
```

To verify the rules, we executed the test below:

```
# From VM LAN:
( kalikali)-[~]
$ echo "pkt1" | nc eden 80
( kalikali)-[~]
$ echo "pkt2" | nc eden 443
( kalikali)-[~]
$ echo "pkt3" | nc eden 22

# On Windows (dns2):
C:\Windows\system32>ncat.exe -l 80
pkt1
^C
C:\Windows\system32>ncat.exe -l 443
pkt2
^C
C:\Windows\system32>ncat.exe -l 22
pkt3

# Linux router rules:
Chain FORWARD (policy DROP 0 packets, 0 bytes)
```

```
 pkts bytes target     prot opt in     out     source           destination
...
    1    60 ACCEPT     tcp  --  *      eth3    192.168.205.0/24  0.0.0.0/0  tcp
        dpt:80
    1    60 ACCEPT     tcp  --  *      eth3    192.168.205.0/24  0.0.0.0/0  tcp
        dpt:443
    1    60 ACCEPT     tcp  --  *      eth3    192.168.205.0/24  0.0.0.0/0  tcp
        dpt:22

Chain POSTROUTING (policy ACCEPT 37 packets, 11192 bytes)
 pkts bytes target       prot opt in   out     source           destination
...
    1    60 MASQUERADE   tcp  --  *    eth3    192.168.205.0/24  0.0.0.0/0  tcp
        dpt:80
    1    60 MASQUERADE   tcp  --  *    eth3    192.168.205.0/24  0.0.0.0/0  tcp
        dpt:443
    1    60 MASQUERADE   tcp  --  *    eth3    192.168.205.0/24  0.0.0.0/0  tcp
        dpt:22
```

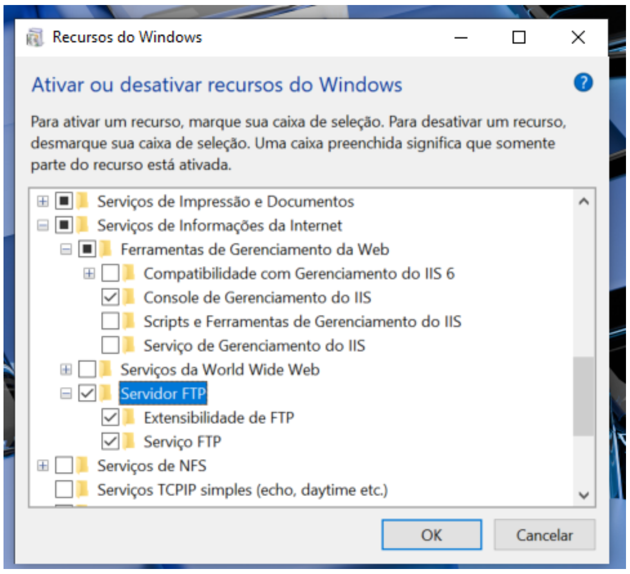
7.3 FTP connections (passive and active modes) to external FTP servers

This section is divided in two parts: external FTP server installation / configuration on Windows and firewall rules.

7.3.1 External FTP server configuration

The following steps detail the configuration of an FTP server on the external Windows host:

1. **Enable the FTP server features in Windows:**
   - Go to *Control Panel > Programs > Turn Windows features on or off.*
   - Enable the following components:
     - *Internet Information Services*
     - *FTP Server*
       - FTP Service
       - FTP Extensibility
     - *Web Management Tools*
       - IIS Management Console


Recursos do Windows

Ativar ou desativar recursos do Windows

Para ativar um recurso, marque sua caixa de seleção. Para desativar um recurso, desmarque sua caixa de seleção. Uma caixa preenchida significa que somente parte do recurso está ativada.

- ■ Serviços de Impressão e Documentos
- ■ Serviços de Informações da Internet
  - ■ Ferramentas de Gerenciamento da Web
    - ☐ Compatibilidade com Gerenciamento do IIS 6
    - ✓ Console de Gerenciamento do IIS
    - ☐ Scripts e Ferramentas de Gerenciamento do IIS
    - ☐ Serviço de Gerenciamento do IIS
  - ☐ Serviços da World Wide Web
  - ✓ Servidor FTP
    - ✓ Extensibilidade de FTP
    - ✓ Serviço FTP
- ☐ Serviços de NFS
- ☐ Serviços TCPIP simples (echo, daytime etc.)

Figure 3: Enable the FTP server features in Windows

2. **Create the FTP site using IIS:**
   - Open the IIS Manager.
   - Click on "Create a new FTP site".
   - Define the following:
     - Site name: `MeuFTP`
     - Physical path: `C:\Users\giovannimaffeo\Documents\UC\2025.1`
   - Configuration:
     - IP address: `192.168.52.6`
     - Port: `21`
     - SSL: `No SSL`
   - Authentication and Authorization:
     - Enable `Basic Authentication`
     - Allow access to: `All users`
     - Permissions: `Read and Write`
   - Finish the configuration.
3. **Configure passive mode for the FTP server:**
   - In IIS Manager, select the created FTP site from the left menu.

- Click on *FTP Firewall Support*.
- Set the *External IP Address* to the FTP server's IP (`192.168.52.6`).
- Define the data channel port range as: `30000-30100`.

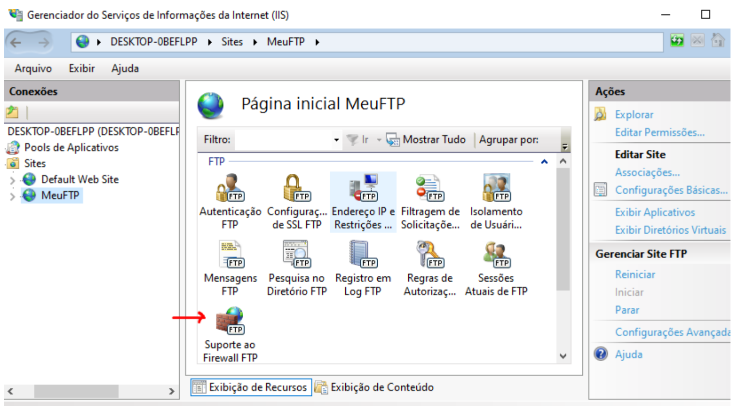


Figure 4: Configure passive mode for the FTP server

4. **Allow FTP ports through Windows Firewall:**

```
netsh advfirewall firewall add rule name="FTP Port 21" dir=in action=allow
     protocol=TCP localport=21
netsh advfirewall firewall add rule name="FTP Passive Ports" dir=in action
    =allow protocol=TCP localport=30000-30100
```

7.3.2 External FTP server firewall rules

To enable both active and passive FTP connections from the internal network (LAN) to an external FTP server, we must create rules that allow forwarding and apply source NAT (MASQUERADE) on the Linux router:

```
sudo iptables -A FORWARD -s 192.168.205.0/24 -p tcp --dport 21 -o eth3 -j ACCEPT
sudo iptables -A FORWARD -s 192.168.205.0/24 -p tcp --dport 30000:30100 -o eth3 -
    j ACCEPT

sudo iptables -t nat -A POSTROUTING -s 192.168.205.0/24 -p tcp --dport 21 -o eth3
     -j MASQUERADE
sudo iptables -t nat -A POSTROUTING -s 192.168.205.0/24 -p tcp --dport
    30000:30100 -o eth3 -j MASQUERADE
```

To verify the rule, we executed the test below:

```
# From VM LAN:
(kalikali)-[~]
$ ftp external_ftp
Connected to external_ftp.
220 Microsoft FTP Service
Name (external_ftp:kali): giovannimaffeo
```

```
331 Password required
Password:
230 User logged in.
Remote system type is Windows_NT.
ftp> dir
229 Entering Extended Passive Mode (|||30001|)
125 Data connection already open; Transfer starting.
04-09-25  11:34AM       <DIR>          Computao  Evolucionria
03-20-25  06:29PM       <DIR>          Engenharia de  Servios
02-16-25  07:02PM       <DIR>          Inteligncia de  Negcio
04-08-25  09:42AM       <DIR>          Reconhecimento de  Padres
03-02-25  02:06PM       <DIR>          Segurana em Tecnologias da  Informao
226 Transfer complete.
ftp> bye
221 Goodbye.

# Linux router rules:
( kalikali)-[~]
$ sudo iptables -L -v -n
sudo iptables -t nat -L -n -v

Chain FORWARD (policy DROP 0 packets, 0 bytes)
...
1    60 ACCEPT     tcp  --  *      eth3    192.168.205.0/24     0.0.0.0/0    tcp
    dpt:21
1    60 ACCEPT     tcp  --  *      eth3    192.168.205.0/24     0.0.0.0/0    tcp
    dpts:30000:30100
...

Chain POSTROUTING (policy ACCEPT 3 packets, 930 bytes)
...
1    60 MASQUERADE  tcp  --  *      eth3    192.168.205.0/24     0.0.0.0/0    tcp
     dpt:21
1    60 MASQUERADE  tcp  --  *      eth3    192.168.205.0/24     0.0.0.0/0    tcp
     dpts:30000:30100
```

8 Intrusion detection and prevention (IDS/IPS)

In this section is explored how we implemented the intrusion detection system using Suricata and also the integration with the IPTables to block the malicious packets that were flagged by the IDS.

8.1 Suricata setup

We started by installing the following applications that were necessary:

```
sudo apt-get install libnetfilter-queue-dev libnetfilter-queue1  \
                libnetfilter-log-dev libnetfilter-log1      \
                libnfnetlink-dev libnfnetlink0

sudo apt-get install suricata
```

We changed the configuration file of Suricata located in /etc/suricata/-suricata.yaml:

```
af-packet:
- interface: eth3
  use-mmap: yes
  tpacket-v3: yes
  mode: inline
  copy-mode: ips
```

Next we also changed the file path for our custom rules, since the Suricata already has one file with rules.

```
default-rule-path: /home/kali/Desktop/rules
```

In the following stage, we updated and reloaded the Suricata using:

```
sudo suricata-update
sudo systemctl restart suricata
```

In order to allow Suricata to block (drop) the connections of the packets that were flagged as attacks, we need to add the following rules in IPTables, otherwise Suricata will only generate the alerts and won't drop the connection regardless the type of rule.

```
sudo iptables -I INPUT -j NFQUEUE --queue-num 0
sudo iptables -I OUTPUT -j NFQUEUE --queue-num 0
sudo iptables -I FORWARD -j NFQUEUE --queue-num 0
```

It is important to note that the simulated attacks cannot be executed against just any server, since we have already defined basic firewall rules in previous sections, reflecting a realistic scenario. Therefore, Suricata should be applied to servers that are exposed in some way to potential threats. To ensure the effectiveness of the tests, the attacks are performed against the `www` server described in Section 2, which is configured to accept incoming connections on ports 443 and 80 from any source, as specified in Subsection 5.7. Even tho attacks, like SQL injection won't do anything, the goal here is to allow a simulation of incoming traffic that contains malicious payload in the body of the request.

8.2 Two types of SQL injection

We decided to perform two different types of attacks, the first one being to list all users in the database and the second an attempt to bypass the login.

The following list shows our rules to detect and block those attacks:

```
drop http any any -> any any (msg:"SQL Injection Detected Attack 1"; content:"
    SELECT * FROM users"; http_client_body; sid:100001;)

drop http any any -> any any (msg:"SQL Injection Detected Login Bypass"; content
    :"admin' OR ''='"; http_client_body; sid:100002;)
```

To test if the rules are valid, we executed the following command:

```
( kalikali)-[~/Desktop/rules]
$ sudo suricata -T suricata.rules
i: suricata: This is Suricata version 7.0.8 RELEASE running in SYSTEM mode
i: suricata: Configuration provided was successfully loaded. Exiting.
```

To start Suricata we performed the following command:

```
sudo suricata -c /etc/suricata/suricata.yaml -q 0 -v -S /home/kali/Desktop/rules/
    suricata.rules
```

We choose to test Suricata rules for traffic that goes to www 5.7, that accepts requests from any source to the server www.

To simulate having a service running in port 80 we open it using netcat with the following command, in the DMZ machine:

```
nc -lp 80
```

To perform the first attack we executed the following command in windows to simulate incoming traffic from other computer to the linux router:

```
curl -X POST www -d "admin' OR ''='" -i
```

And in our linux router we executed:

```
( kalikali)-[~]
$ tail -f /var/log/suricata/fast.log
...
04/20/2025-20:50:51.190527  [Drop] [**] [1:100002:0] SQL Injection Detected Login
     Bypass [**] [Classification: (null)] [Priority: 3] {TCP} 192.168.52.1:30922
    -> 192.168.56.107:80
```

For the second attack, we did the following command in the windows machine:

```
curl -X POST www -d "SELECT * FROM users" -i
```

And we observed in the linux router the second connection:

```
( kalikali)-[~/Desktop/rules]
$ tail -f /var/log/suricata/fast.log
...
04/20/2025-20:49:47.316004  [Drop] [**] [1:100001:0] SQL Injection Detected
    Attack 1 [**] [Classification: (null)] [Priority: 3] {TCP} 192.168.52.1:30877
     -> 192.168.56.107:80
```

8.3 Two types of DoS (Denial of Service) attacks

For the first rule, we simulated a **SYN flood attack**, which is an attack that attempts to exhaust the target's resources by sending a high rate of TCP connection requests without completing the handshake. For this, we created a rule targeting TCP port 443. It triggers when more than 20 SYN packets are sent by the same source within 1 second.

We added the rule below to the file `/etc/suricata/suricata.yaml`:

```
drop tcp any any -> any 80 (msg:"SYN Flood detected on port 443"; flags:S; flow:
    to_server,stateless; threshold:type threshold, track by_src, count 20,
    seconds 1; sid:100003; rev:2;)
```

The test was performed as below:

```
# DMZ (begin www server):
( kalikali)-[~]
$ nc -lp 80

# Attack from Windows:
C:\Users\giovannimaffeo>nping --tcp -p 80 --flags SYN --rate 1000 -c 10000 www

Starting Nping 0.7.95 ( https://nmap.org/nping ) at 2025-04-20 20:51 Horßrio de
    VerÕo de Greenwich
SENT (1.2840s) TCP 192.168.52.1:14661 > 192.168.56.107:80 S ttl=64 id=17819 iplen
    =40  seq=591475376 win=1480
SENT (1.2850s) TCP 192.168.52.1:14661 > 192.168.56.107:80 S ttl=64 id=17819 iplen
    =40  seq=591475376 win=1480
SENT (1.2890s) TCP 192.168.52.1:14661 > 192.168.56.107:80 S ttl=64 id=17819 iplen
    =40  seq=591475376 win=1480
...

# Logs from linux_router Suricata:
( kalikali)-[~/Desktop/rules]
$ tail -f /var/log/suricata/fast.log
...
04/20/2025-20:51:46.040046  [Drop] [**] [1:100003:2] SYN Flood detected on port
    443 [**] [Classification: (null)] [Priority: 3] {TCP} 192.168.52.1:14661 ->
    192.168.56.107:80
04/20/2025-20:51:46.089287  [Drop] [**] [1:100003:2] SYN Flood detected on port
    443 [**] [Classification: (null)] [Priority: 3] {TCP} 192.168.52.1:14661 ->
    192.168.56.107:80
04/20/2025-20:51:46.182067  [Drop] [**] [1:100003:2] SYN Flood detected on port
    443 [**] [Classification: (null)] [Priority: 3] {TCP} 192.168.52.1:14661 ->
    192.168.56.107:80
...
```

For the second rule, we simulated a **UDP flood attack**, which is a type of denial-of-service attack where a large number of UDP packets are sent to random ports on the target machine to consume resources and make the system unresponsive. To detect this, we created a rule targeting UDP packets sent to ports 80 and 443. It triggers when 100 packets are sent by the same source within 5 seconds.

We added the rule below to the file `/etc/suricata/suricata.yaml`:

```
drop udp any any -> any 80 (msg:"DoS UDP flood attempt"; flow:to_server,stateless
    ; threshold:type threshold, track by_src, count 100, seconds 5; sid:100004;
    rev:1;)
```

The test was performed as below:

```
# Attack from Windows:
```

```
nping --udp -p 80 --data-length 32 --rate 1000 -c 10000 www
C:\Users\giovannimaffeo>nping --udp -p 80 --data-length 32 --rate 1000 -c 10000
    www

Starting Nping 0.7.95 ( https://nmap.org/nping ) at 2025-04-20 20:54 Horßrio de
    VerÕo de Greenwich
SENT (1.4450s) UDP 192.168.52.1:53 > 192.168.56.107:80 ttl=64 id=7372 iplen=60
SENT (1.4460s) UDP 192.168.52.1:53 > 192.168.56.107:80 ttl=64 id=7372 iplen=60
...

# Logs from linux_router Suricata:
( kalikali)-[~/Desktop/rules]
$ tail -f /var/log/suricata/fast.log
...
04/20/2025-20:54:19.654264  [Drop] [**] [1:100004:1] DoS UDP flood attempt [**] [
    Classification: (null)] [Priority: 3] {UDP} 192.168.52.1:53 ->
    192.168.56.107:80
04/20/2025-20:54:19.920899  [Drop] [**] [1:100004:1] DoS UDP flood attempt [**] [
    Classification: (null)] [Priority: 3] {UDP} 192.168.52.1:53 ->
    192.168.56.107:80
04/20/2025-20:54:20.161151  [Drop] [**] [1:100004:1] DoS UDP flood attempt [**] [
    Classification: (null)] [Priority: 3] {UDP} 192.168.52.1:53 ->
    192.168.56.107:80
...
```

8.4 OS fingerprinting attempts

The rule to detect and block nmap is the following:

```
drop tcp any any -> any 443 (msg:"BLOCK NMAP -sS simple"; flags:S; flow:to_server
    ,stateless; sid:100005; rev:1;)
```

For the attack, we used the command:

```
C:\Users\giovannimaffeo>nmap -sS -A www
Starting Nmap 7.95 ( https://nmap.org ) at 2025-04-20 20:55 Horßrio de VerÕo de
    Greenwich
Nmap scan report for www (192.168.56.107)
Host is up (0.041s latency).
Not shown: 999 closed tcp ports (reset)
PORT    STATE    SERVICE VERSION
443/tcp filtered https
Too many fingerprints match this host to give specific OS details
Network Distance: 2 hops

TRACEROUTE (using port 993/tcp)
HOP RTT       ADDRESS
1   ...
2   174.00 ms www (192.168.56.107)

OS and Service detection performed. Please report any incorrect results at https
    ://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 16.91 seconds
```

In the log file we can observe that the attack was flagged correctly:

```

( kalikali)-[~/Desktop/rules]
$ tail -f /var/log/suricata/fast.log
```

```
...
04/20/2025-20:55:16.770362  [Drop] [**] [1:100005:1] BLOCK NMAP -sS simple [**] [
    Classification: (null)] [Priority: 3] {TCP} 192.168.52.1:48150 ->
    192.168.56.107:443
04/20/2025-20:55:16.969967  [Drop] [**] [1:100005:1] BLOCK NMAP -sS simple [**] [
    Classification: (null)] [Priority: 3] {TCP} 192.168.52.1:48406 ->
    192.168.56.107:443
04/20/2025-20:55:18.083466  [Drop] [**] [1:100005:1] BLOCK NMAP -sS simple [**] [
    Classification: (null)] [Priority: 3] {TCP} 192.168.52.1:48408 ->
    192.168.56.107:443
...
```

9 PGP submission

This section demonstrates how we encrypted and signed the project ZIP file using PGP keys.

To do this, we accessed the link sent by the professor to copy his public key, which is used to encrypt our ZIP file. It is worth noting that only the professor, who has access to the private key associated with this public key, will be able to decrypt the project.

```
# Access https://flowcrypt.com/pub/bmsousa@ieee.org?show=pubkey
copy the content of the public key
```

Import the professor's public key:

```
gpg --import
paste the content of the public key
control + D
```

Finally, we executed the command to encrypt the project ZIP file with the professor's public key (by specifying the ID using –recipient) and signed it with our private key (using –sign), ensuring authenticity:

```
gpg --sign --encrypt --recipient bmsousa@ieee.org --armor filename
```

Now, the professor can decrypt it by importing our public key to verify the signature and executing the command:

```
gpg --output decrypted_filename --decrypt filename.asc
```

10 Acknowledgments

ChatGPT was utilized in almost all sections with the intention of polishing the English.

11 Conclusion

In conclusion with this work we were able to learn and understand how to implement a router with linux and also how to protect it, by leveraging the IPTables to allow specific connections or drop it. Additionally this project allowed us to learn how to perform attacks and also how to defend from them by integrating an IDS to our setup.